Lab 3
Modified Instructions

EELE 467
Due date: 10/31/2024

Step 1: boot process

Follow Step 1 in the textbook to learn about the SoC FPGA boot process.

Step 2: PuTTY setup

You should already have PuTTY (or a similar terminal emulator program) installed from previous labs. If not, please do so now.

Step 3: Setup TFTP and NFS servers

Follow Step 3 in the textbook, with the following exceptions:

11.1.3.3, Step 5 Skip this step.

11.1.3.3, Step 6 Use 255.255.255.0 for the netmask. You can leave the gateway blank. You can leave the DNS entries as automatic (we aren't connecting to the internet, so DNS will not be used.)

11.1.3.4, Step 3.a Instead of creating an `AudioMini_Passthrough` directory, create an `led-patterns` directory:

```
sudo mkdir -p /srv/tftp/de10nano/led-patterns
```

We won't be using the Audio Mini passthrough project at all this semester (that project is for the second semester).

Step 4: Flashing your SD card

> Stop before Section 11.1.3.6. **Do NOT** flash your SD card with the image provided in Section 11.1.3.6.

If you haven't already, please flash your SD card with Terasic's Linux Console (kernel 4.5) sd card image. Use Balena Etcher to flash your SD card.

Step 5: Change DE10 Nano board switch settings

Follow Step 5 in the textbook.

Step 6: Set the appropriate U-Boot environment variables

Do NOT follow Step 6 in the textbook. Follow the instructions below.

The "developer's boot mode" uses a TFTP server to download the FPGA bitstream, Linux device tree, and Linux kernel; an NFS server is used to host the root filesystem for the ARM processor. U-Boot, the bootloader used by the SoC FPGA, supports downloading files via TFTP, but we need to configure U-Boot accordingly.

Getting to the U-Boot prompt

To set U-Boot environment variables, we need to get into the U-Boot prompt. With your serial console (e.g., PuTTY) open, power on the board and hit any key to stop the boot process and get to the prompt.

Setting environment variables

Type the following at the U-Boot prompt (don't type the lines with #—those are just comments for your understanding):

```
# network setup
setenv serverip 192.168.1.10
setenv ipaddr 192.168.1.11
setenv netmask 255.255.255.0

# ideally, create a random mac address; if you had multiple de10nanos
# on the same network with the same mac address, you would have problems
setenv ethaddr de:ad:be:ef:01:23

# this is the name of the boot script we'll use from the tftp server
setenv netbootscript u-boot.scr

setenv tftpdir de10nano

setenv bootscriptaddr 0x100000

# this command will download the bootscript from our tftp server
# and run it
setenv netboot 'tftp ${bootscriptaddr} ${tftpdir}/bootscripts/${netbootscript}; source
↪  ${bootscriptaddr}'

# true for network booting; false for booting from sd card
setenv usenetboot true

# the command that runs automatically on boot
setenv bootcmd 'if ${usenetboot}; then run netboot; else run mmcbootcmd; fi'
```

```
# the old bootcmd that loads from the sd card
setenv mmcbootcmd 'run callscript; run mmcload; run mmcboot;'
```

Once you've set the environment variables, we need to save them by typing saveenv.

At this point, we should make sure our FPGA can communicate to our Ubuntu virtual machine. With the two devices connected via Ethernet, ping the VM from the U-Boot prompt:

```
ping 192.168.1.10
```

If the ping is not successful, fix the networking issue before moving forward.

Step 7: Setup the LED patterns project and root filesystem

Do NOT follow Step 7 in the textbook. Follow the instructions below.

In this section, we'll put files in our TFTP server so the SoC FPGA can boot our LED patterns project. We'll also set up a root filesystem for the SoC and host it from our VM's NFS server.

TFTP files setup

We'll the put the following files in our TFTP server:

File type	TFTP server location	Description
FPGA bitstream	/srv/tftp/de10nano/led-patterns/	FPGA configuration
device tree blob	/srv/tftp/de10nano/led-patterns/	Tells Linux about the board's hardware
Linux kernel	/srv/tftp/de10nano/kernel/	Linux!
U-Boot script	/srv/tftp/de10nano/bootscripts/	Tells U-Boot how to setup and boot our project

Copying pre-existing files

1. Create a directory for your LED patterns project:

   ```
   mkdir -p /srv/tftp/de10nano/led-patterns
   ```

2. Copy the rbf file (FPGA bitstream) for your LED patterns project to `/srv/tftp/de10nano/led-patterns/led-patterns.rbf`.

3. Copy the .dtb file from your SD card's FAT32 partition to `/srv/tftp/de10nano/led-patterns/led-patterns.dtb`.

4. Copy the `zImage` from your SD card's FAT32 partition to `/srv/tftp/de10nano/kernel/`.

Create the U-Boot script

1. Create `/srv/tftp/de10nano/bootscripts/led-patterns.script` with the following contents:

```
# file directories
setenv tftpkerneldir ${tftpdir}/kernel
setenv tftpprojectdir ${tftpdir}/led-patterns
setenv nfsrootdir /srv/nfs/de10nano/ubuntu-rootfs

# kernel bootargs
setenv bootargs console=ttyS0,115200 ip=${ipaddr} root=/dev/nfs rw
↪  nfsroot=${serverip}:${nfsrootdir},v4,tcp nfsrootdebug earlyprintk=serial

# file names
setenv fpgaimage led-patterns.rbf
setenv dtbimage led-patterns.dtb
setenv bootimage zImage

# memory addresses where files get loaded into
setenv fpgadata 0x2000000
setenv fpgadatasize 0x700000
setenv dtbaddr 0x00000100
setenv kerneladdr 0x8000

# commands to get files, configure the fpga, and load the kernel
setenv getfpgadata 'tftp ${fpgadata} ${tftpprojectdir}/${fpgaimage}'
setenv getdtb 'tftp ${dtbaddr} ${tftpprojectdir}/${dtbimage}'
setenv getkernel 'tftp ${kerneladdr} ${tftpkerneldir}/${bootimage}'
setenv loadfpga 'fpga load 0 ${fpgadata} ${fpgadatasize}'

# get all of the files and boot the device
run getfpgadata;
run loadfpga;
run getdtb;
run getkernel;
run bridge_enable_handoff;
bootz ${kerneladdr} - ${dtbaddr}
```

2. `led-patterns.script` needs to be converted into a U-Boot image with `mkimage`.

   (a) Install `mkimage`:

   ```
   sudo apt install u-boot-tools
   ```

   (b) Create the image:

   ```
   mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "LED patterns
   ↪  bootscript" -d led-patterns.script led-patterns.scr
   ```

3. Earlier, we told U-Boot to download `u-boot.scr`. To make changing projects easy, we'll make `u-boot.scr` a symbolic link to the U-Boot script we want to run; this is easier than dropping into the U-Boot console and changing the `bootscript` variable every time we want to boot a different project. Create a symbolic link to `led-patterns.scr`:

```
ln -s led-patterns.scr u-boot.scr
```

Root filesystem setup

Our NFS server will serve the root filesystem for the ARM processor.

1. Download the minimal armhf Ubuntu root filesystem we've created for you.
2. Put the `ubuntu-rootfs.tar.gz` file in `/srv/nfs/de10nano/`.
3. Extract the root filesystem:

   ```
   sudo tar -xpvf ubuntu-rootfs.tar.gz
   ```

4. Remove the tar file:

   ```
   sudo rm ubuntu-rootfs.tar.gz
   ```

> In the root filesystem, we created the following user:
>   username: `soc`
>   password: `fpga`
> This will be how you log into Linux on the ARM processor.

Step 8: Setup the cross-compilation toolchain

You should already have the cross compiler installed and working from Labs 7 and 8. If not, please do so now.

Step 9: Manually cross-compile Hello World

Follow Step 9 in the textbook. Although we've already cross-compiled code for Labs 7 and 8, we want to make sure our NFS server is working.

> Instead of copying your executable files to `/srv/nfs/de10nano/ubuntu-rootfs/root/`, copy them to `/srv/nfs/de10nano/ubuntu-rootfs/home/soc/`, as that is the home folder for our soc user. You likely do not need to use `sudo` to copy files to the soc home directory.

> Moving forward, we no longer need to use static linking when cross-compiling our code. The root filesystem we're hosting via the NFS server is Ubuntu 24.04, so it has a gcc compiler and `glibc` library that are compatible with the Ubuntu version running on your VM.

(optional) Step 10: Cross-compile Hello World using a Makefile

> This step is optional. If you want to use a Makefile for compiling code in the future, you can follow this step.

Follow Step 10 in the textbook. The `Makefile` and `arm-env.sh` files have been put in the `utils/` folder in your repository—look for a pull request.

Deliverables

- Demo the same demonstrations listed in the textbook.
- Include a screenshot of your hello world program running on the ARM CPU in your `lab3.md` file.